Raymond Salzmann

EEL3744C

Lab 6

Pre-Lab Questions

1. What is the highest speed of communication the IMU can handle?

   10MHz

2. In which order should you transmit data to the LSM330? LSB first or MSB first?

   You should transmit data LSB first

3. How are the accelerometer and gyroscope enabled?

   After writing all of the functions that allow us to access the values of the accelerometer/gyroscope, we create interrupts to show when to read these values. We enable axis values and output rates.

4. When using SPI, why do we have to write data in order to be able to read data?

   We need to write to the slave to tell it that we want to read data.

5. Why is it a good idea to modify global flag variables inside ISRs instead of doing everything inside of them?

   Having interrupts that take many clock cycles to complete could cause weird things to happen with our processor.

Problems Encountered

I had a lot of problems with Part F and my data not being shown to the graph even though there were no apparent issues with my code, I tried removing my header files besides LSM330 and adding all of the functions I had written to my main file.

Future Work/Applications

If I had more time to work on this lab I would tidy up my code and restore the header files that I initially did. Try not to freak out last second.

Pre-Lab

Part A:

Pseudocode:

```
1. Initialize the clock to 32MHz
2. Initialize the SPI
```

Part B:

Pseudocode:

```
1. Use the data register with SPI write, much like outchar to create the function
2. Much like inchar use the return on the data register for spiread
```

Program Code:

```
/* Lab 6 Part B
 * Name: Raymond Salzmann
 * Section #: 2B04
 * TA Name: Keith Fitzgerald
 * Description: SPI Communication Testing
 */

#include <avr/io.h>
#include <avr/interrupt.h>

void CLK_INIT(void);
uint8_t SPI_SETUP(void);
uint8_t spiRead(void);
uint8_t spiWrite(uint8_t data);


int main(void)
{
   CLK_INIT(); //initialize clock to 32MHz
   SPI_SETUP(); //initialize the SPI

       while (1){

       spiWrite(0x53); //constantly write 53 using SPiwrite

       }
}

void CLK_INIT(void){
       OSC_CTRL = 0x02;

       while(!(OSC_STATUS & 0x01));

       CPU_CCP = 0xD8;

       CLK_CTRL = 0x01;

       return;
}

uint8_t SPI_SETUP(void){

       PORTF_DIRSET = 0xB0;

       PORTF_DIRCLR = 0x40;

       SPIF_CTRL =  0x5C;
```

```
        return;
}

uint8_t spiWrite(uint8_t data){

        SPIF_DATA = data; //put data into the data register

        while(!(SPIF_STATUS |= 0x80)); //wait until cleared

        return SPIF_DATA; //return data register
}

uint8_t spiRead(void){

        return spiWrite(0xFF);
}
```

Part C:

Pseudocode:

1. In the readLSM function, set up the PORTF_OUT register to properly enable the accelerometer
2. Also in this function call spi write but make sure that the MS bit is not set
3. For write LSM, set up the PORTF_OUT register to properly enable the accelerometer
4. Also in this function call spi write to write two bytes of data back to back

Part D:

Pseudocode:

1. Initialize the accelerometer
2. Set up the interrupt for the accelerometer so that the interrupt depends on pin 7 of port C

Part E:

Pseudocode:

1. Initialize the USARTD0
2. Set up the USART so that it can transmit and receive data continuously
3. Set up the USART asynchronously

Part F:

Pseudocode:

1. Carry over all of the functions from the previous parts into this part
2. Set up the main function to call these functions

3. Set up so that Protocol_SEL is zero
4. Set up so that when the ISR is triggered by Pin 7 on Port C, we set the value of accelDataReady to 0x01
5. This causes the main to output the data from the accelerometer
6. Debug the processor with the Graph from Data Visualization to see the Real-time Data Plotting values

Program Code:

```
/* Lab 6 Part F
 * Name: Raymond Salzmann
 *Section #: 2B04
 * Description: Real Time Data Plotting
 * TA Name: Keith Fitzgerald
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include "LSM330.h"
#include "usart.h"

void   CLK_INIT(void);
uint8_t spiWrite(uint8_t data);
uint8_t spiRead(void);
uint8_t spi_init(void);
uint8_t LSM_read(uint8_t byte1);
uint8_t LSM_write(uint8_t byte1, uint8_t byte2);
uint8_t accel_init(void);

volatile uint8_t accelDataReady = 0;

int main(void)
{
       volatile uint8_t accel_x_low;            //initialize variables for accelerometer
data
       volatile uint8_t accel_x_high;

       volatile uint8_t accel_y_low;
       volatile uint8_t accel_y_high;

       volatile uint8_t accel_z_low;
       volatile uint8_t accel_z_high;

       CLK_INIT();   //initialize clock
       PORTF_DIRSET = 0x18;
       PORTF_OUTSET = 0x18;

       PORTA_DIRSET = 0x10;
       PORTA_OUTCLR = 0x10;

       spi_init(); //initialize spi

       LSM_write(CTRL_REG4_A, 0x01);

       accel_init();
       USART_INIT(); //initialize the baud rate and usart
```

```
        while(1)
        {
                if(accelDataReady == 1)
                {
                        accel_x_low  = LSM_read(OUT_X_L_A);             //get values for the
high and low of x,y, and z
                        accel_x_high = LSM_read(OUT_X_H_A);
                        accel_y_low  = LSM_read(OUT_Y_L_A);
                        accel_y_high = LSM_read(OUT_Y_H_A);
                        accel_z_low  = LSM_read(OUT_Z_L_A);
                        accel_z_high = LSM_read(OUT_Z_H_A);

                        accelDataReady = 0;                                     //clear
interrupt

                        //output the data

        DISPLAY_DATA(accel_x_low,accel_x_high,accel_y_low,accel_y_high,accel_z_low,accel_z
_high);
                }
        }
        return;
}

uint8_t accel_init(void)
{
        PORTC_PIN7CTRL = 0x01;  //rising edge interrupt port c pin 7

        PORTC_DIRCLR = 0x80; //set pin 7 as input

        PORTC_INT0MASK = 0x80; //turn on interrupt for pin 7 port c

        PORTC_INTCTRL = 0x01; //set up for low level interrupt

        LSM_write(CTRL_REG4_A, 0xC8);           //write to LSM registers

        LSM_write(CTRL_REG5_A, 0x97);

        PMIC_CTRL = 0x01; //enable all interrupts

        sei();

        return;
}

uint8_t LSM_read(uint8_t addr)
{
        PORTF_OUTCLR = 0x08;

        PORTF_OUTSET = 0x04;

        spiWrite((addr & 0x3F) | 0x80);

        uint8_t read_data = spiRead();

        PORTF_OUTSET = 0x08;
        return read_data;
}

uint8_t LSM_write(uint8_t addr, uint8_t data)
{
        PORTF_OUTCLR = 0x08;

        PORTF_OUTSET = 0x04;

        spiWrite(addr);       //spi write(byte1)

        spiWrite(data);       //spi_write(byte2)

        PORTF_OUTSET = 0x08; //slave select signal high

        return;
}

void CLK_INIT(void)
{

        OSC_CTRL                    = 0x02;

        while(!(OSC_STATUS & 0x02));

        CPU_CCP                            = 0xD8;


        CLK_CTRL                    = 0x01;

}

uint8_t spiWrite(uint8_t data)
{
        SPIF_DATA = data;

        while(SPIF_STATUS != 0x80);

        return SPIF_DATA;
}

uint8_t spiRead(void)
{
        return spiWrite(0xFF);
}

uint8_t spi_init(void)
{
        PORTF_DIRSET =       0xBC; //set direction for spi

        SPIF_CTRL =          0x50; //set control for spi

        SPIF_INTCTRL = 0x00; //turn off interrupts for spi
        return;
}

ISR(PORTC_INT0_vect)
{
        accelDataReady = 1; //in interrupt tell that data is ready
        PORTC_INTFLAGS = 0x01; //clear flag
        return;
}
```

Part G:

Pseudocode:

```
1. Take the code from Part F and implement a system where it converts the values received from the accelerometer
2. From these converted values, control the timer counter registers for the duty cycles of the LEDs based on the values from the accelerometer
```

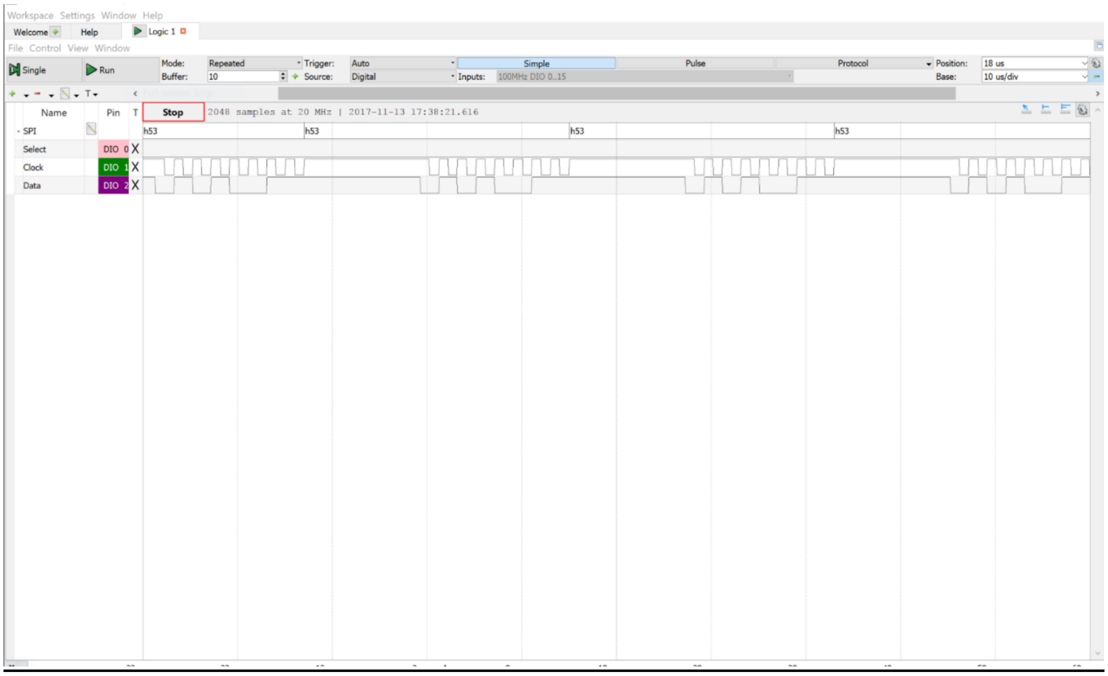
Program Code:

```
* Lab 6 Part G
 * Name: Raymond Salzmann
 *Section #: 2B04
 * Description: Real Time Data Plotting
 * TA Name: Keith Fitzgerald
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include "LSM330.h"
#include "usart.h"

void   CLK_INIT(void);
uint8_t spiWrite(uint8_t data);
uint8_t spiRead(void);
uint8_t spi_init(void);
uint8_t LSM_read(uint8_t byte1);
uint8_t LSM_write(uint8_t byte1, uint8_t byte2);
uint8_t accel_init(void);
void PWM_init(void);

volatile uint8_t accelReady = 0;

int main(void)
{
       volatile uint8_t accel_x_low;            //initialize variables for accelerometer
data
       volatile uint8_t accel_x_high;

       volatile uint8_t accel_y_low;
       volatile uint8_t accel_y_high;

       volatile uint8_t accel_z_low;
       volatile uint8_t accel_z_high;

       CLK_INIT();   //initialize clock

       PORTF_DIRSET = 0x18;
       PORTF_OUTSET = 0x18;

       PWM_init();
       PORTA_DIRSET = 0x10;
       PORTA_OUTCLR = 0x10;

       spi_init(); //initialize spi

       LSM_write(CTRL_REG4_A, 0x01);

       accel_init(); //acceleration initialization
       USART_INIT(); //initialize the baud rate and usart

       while(1)
       {
              if(accelReady == 1)
              {
                     accel_x_low  = LSM_read(OUT_X_L_A);             //get values for the
high and low of x,y, and z
                     accel_x_high = LSM_read(OUT_X_H_A);

                     accel_y_low  = LSM_read(OUT_Y_L_A);
                     accel_y_high = LSM_read(OUT_Y_H_A);

                     accel_z_low  = LSM_read(OUT_Z_L_A);
                     accel_z_high = LSM_read(OUT_Z_H_A);

                     accelReady = 0;                                        //clear
interrupt

                     uint16_t xval = abs( (accel_x_high << 8) | accel_x_low); //take
absolute value as said in lab doc
                     uint16_t yval = abs( (accel_y_high << 8) | accel_y_low); //take
absolute value as said in lab doc
                     uint16_t zval = abs( (accel_z_high << 8) | accel_z_low); //take
absolute value as said in lab doc



                     TCD0_CCA = xval; //red
                     TCD0_CCB = yval; //green
                     TCD0_CCC = zval; //blue
              }
       }
       return;
}

uint8_t accel_init(void)
{
       PORTC_PIN7CTRL = 0x01;  //rising edge interrupt port c pin 7

       PORTC_DIRCLR = 0x80; //set pin 7 as output

       PORTC_INT0MASK = 0x80; //setup pin 7 as an interrupt for port c

       PORTC_INTCTRL = 0x01; //low level interrupt

       LSM_write(CTRL_REG4_A, 0xC8);             //write to LSM registers

       LSM_write(CTRL_REG5_A, 0x97);
```

```
        PMIC_CTRL = 0x01; //enable all interrupts
        sei();

        return;
}

uint8_t LSM_read(uint8_t addr)
{

        PORTF_OUTCLR = 0x08;

        PORTF_OUTSET = 0x04;

        spiWrite((addr & 0x3F) | 0x80);

        uint8_t read_data = spiRead();

        PORTF_OUTSET = 0x08;

        return read_data;
}

uint8_t LSM_write(uint8_t addr, uint8_t data)
{

        PORTF_OUTCLR = 0x08;

        PORTF_OUTSET = 0x04;

        spiWrite(addr);       //spi write(byte1)

        spiWrite(data);       //spi_write(byte2)

        PORTF_OUTSET = 0x08; //slave select signal high

        return;
}

void CLK_INIT(void)
{
        //set up clock for 32MHz
        OSC_CTRL                    = 0x02;

        while(!(OSC_STATUS & 0x02));


        CPU_CCP                            = 0xD8;


        CLK_CTRL                    = 0x01;

}

uint8_t spiWrite(uint8_t data)
{
        SPIF_DATA = data; //write to the data register

        while(SPIF_STATUS != 0x80);
        return SPIF_DATA;
}

uint8_t spiRead(void)
{
        return spiWrite(0xFF);
}

uint8_t spi_init(void)
{
        PORTF_DIRSET =       0xBC; //set portf direction for spi

        SPIF_CTRL =          0x50; //set control for spi

        SPIF_INTCTRL = 0x00; //turn off interrupts

        return;
}

ISR(PORTC_INT0_vect)
{
        accelReady = 1;
        PORTC_INTFLAGS = 0x01; //clear flag
        return;
}

void PWM_init(void)
{       //initialize the PWM from lab 3

        PORTD_REMAP          = 0x07;             //;remap the three pins to use later

        TCD0_CTRLA = 0x03;

        TCD0_CTRLB           = 0x73;             //;make sure the led pins are good

        PORTD_DIRSET  = 0b01110000; //;set all the leds to be outputs

        PORTD_PIN6CTRL       = 0x40;             //;invert the pins to make writing easier

        PORTD_PIN5CTRL = 0x40;

        PORTD_PIN4CTRL = 0x40;

        PMIC_CTRL |=  0x01;

        TCD0_PER = 0x3FFF;

        TCD0_CNT = 0x3FFF;

}
```

Appendix

Part B Screenshot:

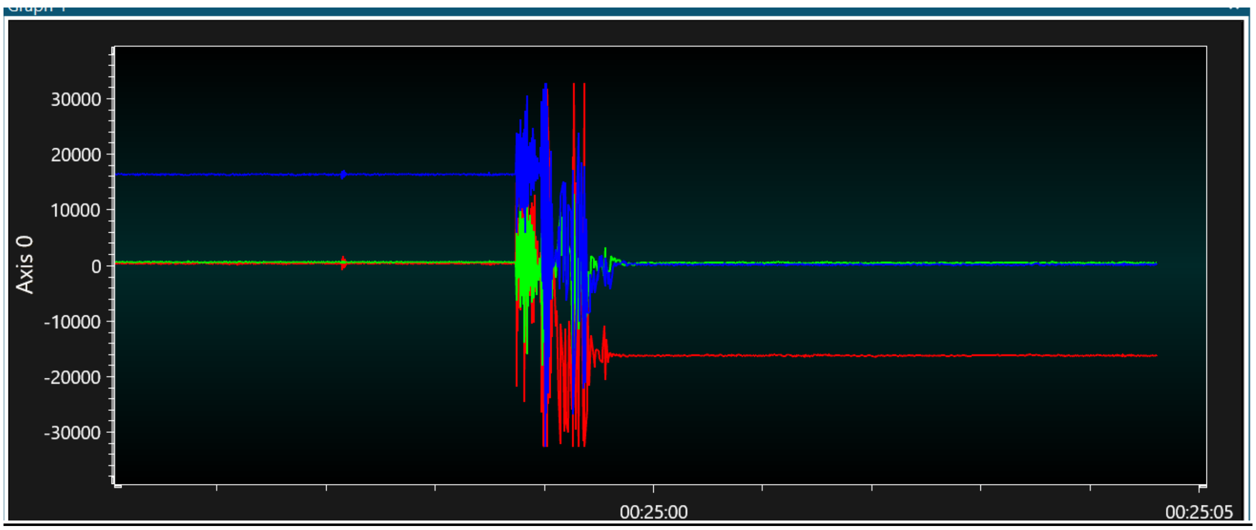


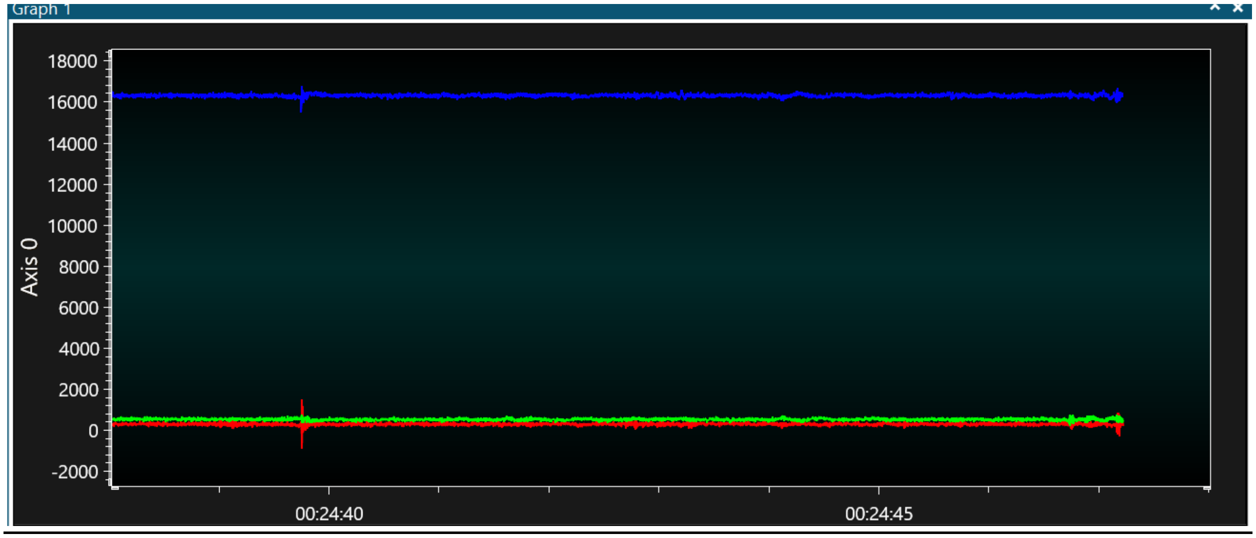
Mode: Repeated
Buffer: 10
Trigger: Auto
Source: Digital
Inputs: 100MHz DIO 0..15
Position: 18 us
Base: 10 us/div

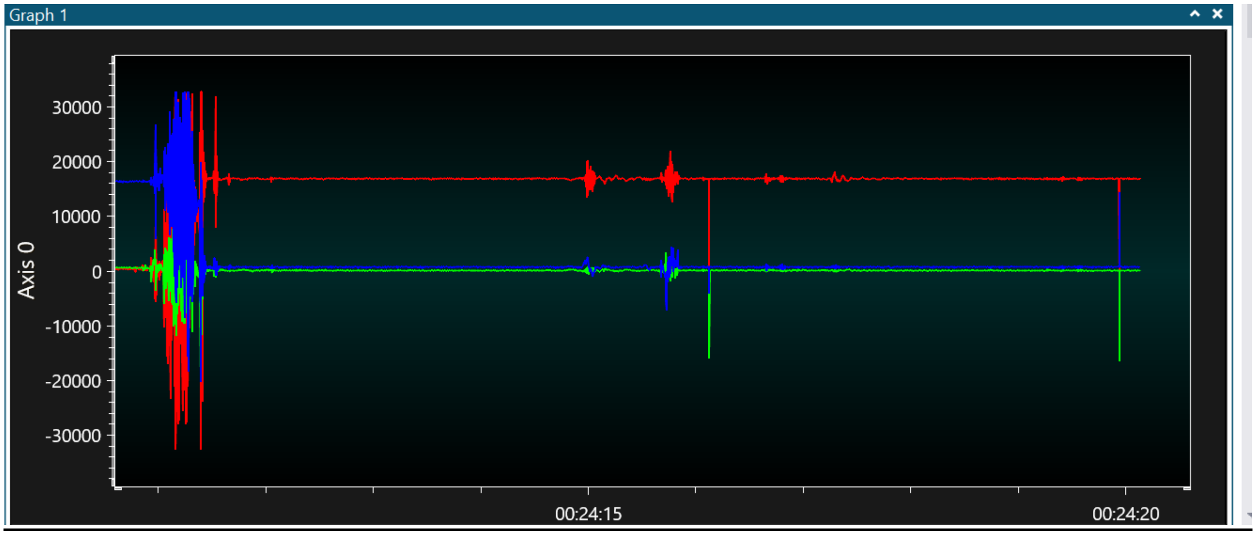
Name	Pin
Select	DIO 0
Clock	DIO 1
Data	DIO 2

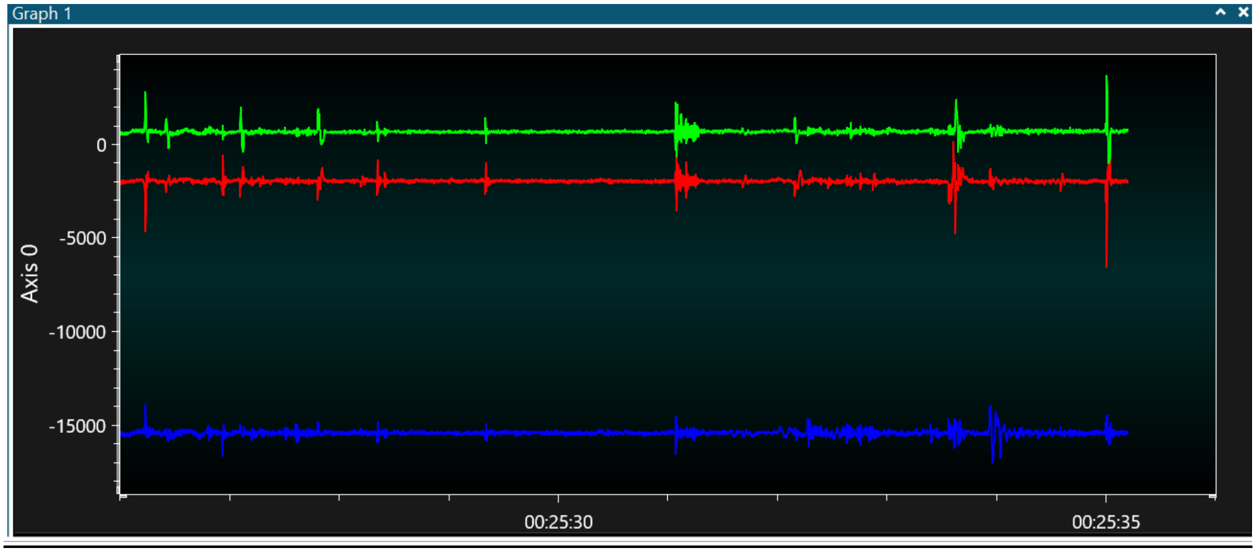
```
2048 samples at 20 MHz | 2017-11-13 17:38:21.616
```

<u>Accelerometer Graph (One from each side):</u>




Graph 1
Axis 0
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
-2000
00:24:40
00:24:45


Graph 1
Axis 0
30000
20000
10000
0
-10000
-20000
-30000
00:24:15
00:24:20


Graph 1
Axis 0
0
-5000
-10000
-15000
00:25:30
00:25:35